Application Report

Integrating Virtual DRM Between VISION SDK and PSDK on Jacinto6 SOC

Fredy Zhang, Joe Shen, and Peter Li

ABSTRACT

The Direct Rendering Manager (DRM) is widely used by user-space graphic stacks. In a setup where A15 is not allowed to access DSS, a working omapdrm driver is not available to display content. Virtual DRM is a driver framework to expose DRM devices to Linux® userspace in such a setup, therefore, enabling Linux DRM applications to continue functioning.

Project collateral and source code discussed in this document can be downloaded from the following URL: https://www.ti.com/lit/zip/spracx5.

Table of Contents

Trademarks

1 Introduction

The DRM/KMS framework is dedicated to the management of the display, graphic and composition subsystems, as shown in Figure 1-1.

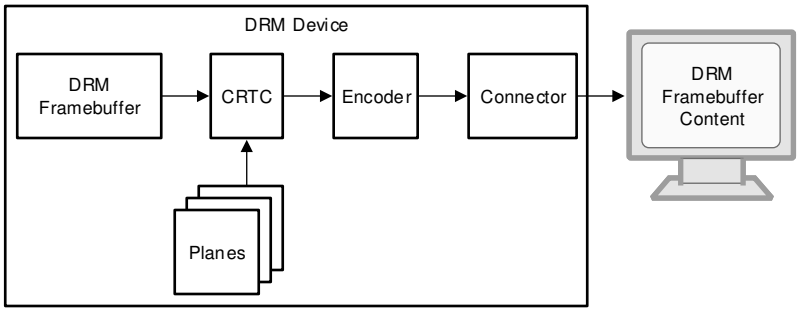

Figure 1-1. DRM/KMS Architecture

With the help of other Linux multimedia frameworks and applications. the DRM/KMS framework is typically used:

- To compose animated contents taking advantages of the hardware acceleration.
- To control both display interfaces and external displays including their settings (resolution, frequencies, multi-screen, and so forth).
- To display this animated content on display panels or HDMI outputs.

DRM device: Responsible for aggregating the other components. Device exposed to the user space (handles all user-space requests.)

DRM Framebuffer: This is a standard object storing information about the content to be displayed.

CRTC: CRTC stands for CRT Controller, it scans out frame buffer content to one or more displays and update the frame buffer.

Planes: A plane is an image layer

Encoder: Responsible for converting a frame into the appropriate format to be transmitted through the connector.

Connector: Represent a display connector (HDMI, DP, VGA, DVI, and so forth), transmit the signals to the display. Detect display connection/removal. Expose display supported modes.

In vision SDK Linux, DSS is controlled by software running on IPU. As a result, omapdrm needs to be disabled, and Linux based DRM applications cease to function properly as there is no DRM device capable of modesetting (displaying content). A virtual DRM framework was introduced to create multiple DRM devices capable of modesetting and expose them to User space.

Using the vDRM framework, on the one hand, vDRM support the Linux display. On the other hand, M4 can control the DSS hardware. So when the M4 starting, it can display content by M4.

Table 1-1 shows a DRM comparison of the DRM of PSDKLA and VISION SDK.

Table 1-1. DRM Comparison of PSDKLA and VISION SDK

Type	PSDKLA	VISION SDK
DRM	DRM	Virtual DRM
DSS	Controlled by A15 (Linux)	Controlled by M4 (RTOS)
Omapdrm support	YES	NO
Fb0	YES	NO

1.1 Standard DRM Framework

Normally, A IVI system may include the three parts: HMI plane, Logo plane and RVC video plane. A cluster system may include the two parts: HMI planes and animation planes.

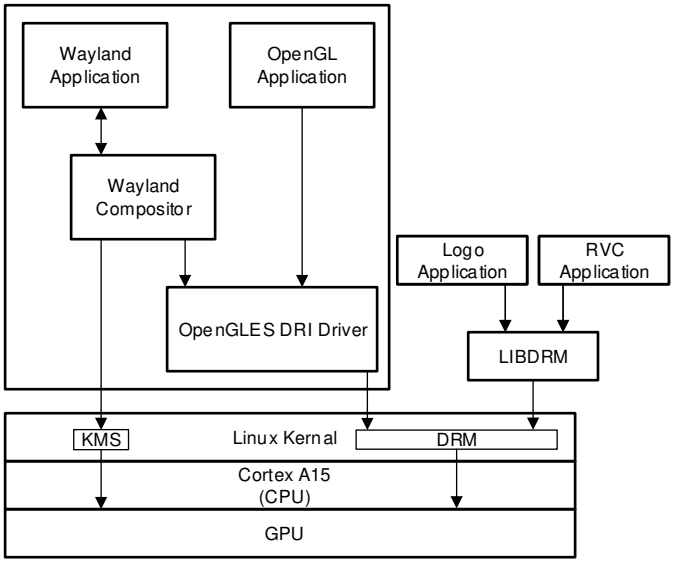

Figure 1-2. The Standard Framework

As shown in Figure 1-2, the standard framework, the Direct Rendering Manager (DRM) resides in kernel space, so user-space programs must use kernel system calls to request its services. A library called *libdrm* was created to facilitate the interface of user-space programs with the DRM subsystem. This library is merely a wrapper that provides a function written in C for every ioctl of the DRM API, as well as constants, structures and other helper elements.

1.2 vDRM-Based Framework

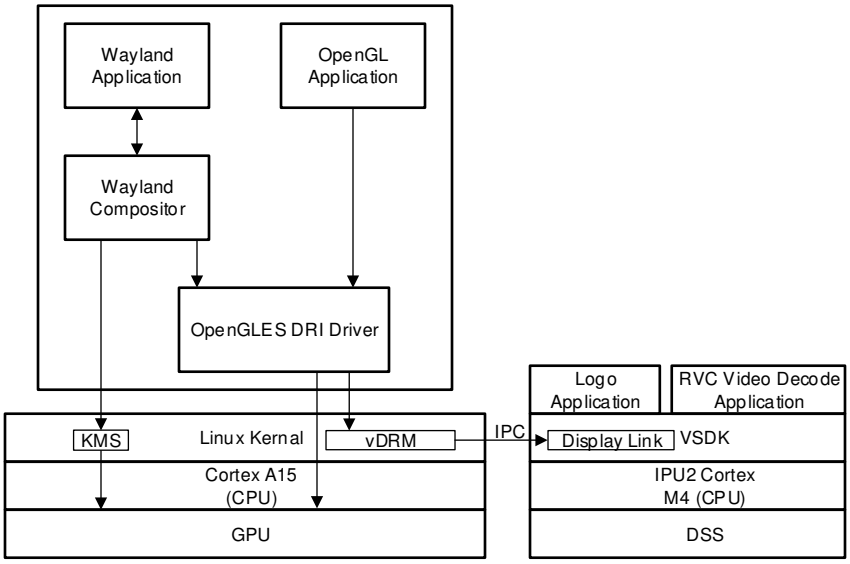

Figure 1-3. The vDRM Framework

In this framework, the omapdrm needs to be disabled in Linux as DSS is controlled by software running on IPU. The Linux application that based DRM will not work as there is no DRM device capable of mode setting (displaying content).

Virtual DRM can create multiple DRM devices capable of modesetting and expose them to user space. Each DRM device can contain multiple DRM connectors, and each connector can be configured to expose a predefined resolution and frame rate. Each DRM connector internally creates a DRM encoder, a DRM plane (primary) and a DRM CRTC, which are needed by DRM APIs to function properly.

Additionally, each DRM device creates a vdrm-controller device which can be opened by Linux applications to read the buffers submitted by DRM applications. Vision SDK can run a chain (usecase) with multiple instances of dispDistSrcLink, where each link reads a vdrm-controller device to obtain buffers submitted by a DRM application to a particular CRTC in a virtual DRM device.

Linux applications can continue to call DRM APIs to display a DRM Frame buffer on a DRM CRTC, even when the vision SDK application / chain is not running, or the running chain does not contain the dispDistSrcLink associated with the CRTC.

From VISION SDK 0304, vDRM framework support in SDK.

2 Display Content Based vDRM on Linux

For Linux to display content, it needs to run dispDistSrcLink and DisplayLink on M4 so that the Linux application buffer can transfer to the M4 for display. Use the following steps to run the Linux application:

1. Run apps.out on Linux terminal.
2. # press 1 to select the single camera usecase.
3. # press 8 to select the dispdist usecase.
4. ctrl+z back to Linux terminal.
5. Run Linux application.

SDK also provide the fast boot method to run the usecases. Use the following steps to run the usecase:

1. Change the filesystem script: init-demo.sh (/home/root) :
   a. From LAUNCH_DEMO=0 to LAUNCH_DEMO=1.
2. Reboot the board, the system can run apps.out automatically.

3 Multimedia Support Based vDRM on Linux

Due to the IVI/cluster/ADAS project has a specific requirement for boot time. Different with the standard DRM display framework. The vDRM framework can adjust the multicore architecture and improve the boot time performance.

For PSDKLA + VISION-SDK architecture, we often use early boot late-attach. Figure 3-1 shows the boot flow.

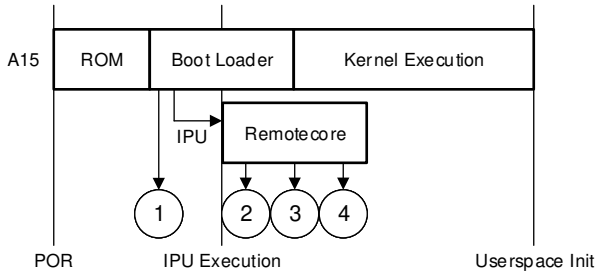

Figure 3-1. The Boot Flow

For some application that based omapdrm will not work with virtual DRM. So we need to adjust the vDRM requirement. Here are some examples: Gstreamer/viddec3test/modetest/kmscube. Those binaries can download from the attachment.

3.1 Gstreamer

For gstreamer, it will not work with prebuilt gstreamer drm allocator. It needs Gstreamer OmapDRM allocator support. We provide a patch to support this feature.

Use the following steps in order to run the decode feature:

1. Enable IPUMM in VISON SDK.

   PROCESSOR_SDK_VISION_03_05_00_00/vision_sdk/apps/configs/tda2xx_evm_linux_all/cfg.mk

```
# Both IVAHD_INCLUDE & IPUMM_INCLUDE should not be set to "yes"
# Only one should be enabled to avoid IVA-HD resource conflict
IPUMM_INCLUDE=yes
IVAHD_INCLUDE=no
```

2. Boot the board, run the command shown below:

```
root@dra7xx-evm# cd /opt/vision_sdk
root@dra7xx-evm# ./vision_sdk_load.sh
root@dra7xx-evm# ./apps.out
# press 1 to select : single camera usecases
# press 8 to select :  dispDistSrc -> display usecase
root@dra7xx-evm#  gst-launch-1.0 playbin uri=file:///home/root/test.mp4 video-sink=waylandsink
```

3.2 viddec3test

If IPUMM was included in the build, you can run any one of the following commands on the ssh terminal to validate multimedia.

```
root@dra7xx-evm# viddec3test -w 640x480 --fps 24 /usr/share/ti/video/TearOfSteel-Short-1920x800.mov
```

3.3 modetest

- The tool **modetest** provided by the libdrm library is useful to:
  - List all display capabilities: CRTCs, encoders and connectors (DSI, DPI, HDMI, ...), planes, modes, and so forth
  - Perform basic tests: display a test pattern, display 2 layers, perform a vsync test
  - Specify the video mode: resolution and refresh rate

For vDRM framework, you can run the following to see DRM related information for each card:

```
root@dra7xx-evm# modetest -n /dev/dri/card1
root@dra7xx-evm# modetest -n /dev/dri/card1
root@dra7xx-evm# modetest -n /dev/dri/card2
```

3.4 kmscube

kmscube is a little demonstration program for how to drive bare metal graphics without a compositor like X11, wayland or similar, using DRM/KMS (kernel mode setting), GBM (graphics buffer manager) and EGL for rendering content using OpenGL or OpenGL ES.

For vDRM framework, please use the below command to run kmscube. If you want to run the kmscube, make sure the weston is stopped (weston is running by default on /dev/dri/card0).

```
root@dra7xx-evm#  kmscube -d /dev/dri/card0
```

4 Display Weston-Based Application

Weston is running by default on /dev/dri/card0. Example for user use vDRM to display user app.

Check the file: /etc/powervr.ini.

```
[default]
#WindowSystem=libpvrws_WAYLAND.so

[weston]
DbmDriverName=vdrm

[kmscube]
DbmDriverName=vdrm
GbmNumBuffers=5
```

Make sure the Weston is configured in this file. Then, you can check the Weston status and run the Weston application.

5 Display EGL-Based Application

Configure the EGL based application, see the file: /etc/powervr.ini.

```
[default]
#WindowSystem=libpvrws_WAYLAND.so

[weston]
DbmDriverName=vdrm

[kmscube]
DbmDriverName=vdrm
GbmNumBuffers=5
```

If you want to add the user app, add your app configuration as shown below:

```
[user_app]
DbmDriverName=vdrm
GbmNumBuffers=5
```

Except the configuration, you also need to configure the device name in your app. A patch (0001-add-flag-for-device-node-name.patch) is provided for your reference.

6 Interactive Display Across PSDKLA and VISION-SDK

You want to display Linux and VISION-SDK applications for some use cases, which are typically using a different pipe lane. You can change the Alpha and Zorder setting in those use cases.

6.1 ALPHA Setting

Address Offset	0x0000 0074		
Physical Address	0x5800 1074	**Instance**	DISPC
Description	The register defines the global alpha value for the graphics and three video pipelines. Shadow register, updated on VFP start period of primary LCD or VFP start period of the third LCD or VFP start period of the secondary LCD or VFP start period of the third LCD or EVSYNC or when DISPC_CONTROL2.GOWB is set to 1 by software and current WB frame is finished (no more data in the write-back pipeline). The synchronization event is defined based on the output using the pipeline: primary LCD, secondary LCD, third LCD, TV output or write-back to the memory for each bit field depending on the association of the each pipeline with the primary LCD, secondary LCD or TV output.		
Type	RW		

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
VID3GLOBALALPHA	VID2GLOBALALPHA	VID1GLOBALALPHA	GFXGLOBALALPHA

Bits	Field Name	Description	Type	Reset
31:24	VID3GLOBALALPHA	Global alpha value from 0 to 255. 0 corresponds to fully transparent and 255 to fully opaque.	RW	0xFF
23:16	VID2GLOBALALPHA	Global alpha value from 0 to 255. 0 corresponds to fully transparent and 255 to fully opaque.	RW	0xFF
15:8	VID1GLOBALALPHA	Global alpha value from 0 to 255. 0 corresponds to fully transparent and 255 to fully opaque.	RW	0xFF
7:0	GFXGLOBALALPHA	Global alpha value from 0 to 255. 0 corresponds to fully transparent and 255 to fully opaque.	RW	0xFF

Figure 6-1. Alpha Setting

6.2 ZORDER Setting: DISPC_xxx_ATTRIBUTES[26-27]

27:26	ZORDER	Z-Order defining the priority of the layer compared to others when overlaying. It is software responsibility to ensure that each layer connected to the same overlay manager has a different z-order value. If bit 25 is set to 0, the ZORDER bit field is ignored and replaced by the value 0. 0x0: Z-order 0: layer above solid background color and below layer with higher Z-order values. 0x1: Z-order 1: layer above layer with z-order value of 0 and below layers with z-order values of 2 and 3 0x3: Z-order 3: layer above all the other layers 0x2: Z-order 2: layer above layers with z-order value of 0 and 1 and below layer with z-order value of 3	RW	0x0

Figure 6-2. ZORDER Setting: DISPC_xxx_ATTRIBUTES[26-27]

7 Dual-Display Demo

The dual-display demo needs two screens to display different content: one screen for cluster and another screen for surround view.

Attachment provides a dual Display-Demo use case.

8 Build Linux Vision SDK File System

From Vision-SDK 3.4, there are changes in the file-system build to enable a smaller file-system. While the file-system provided as part of the Processor-SDK Linux automotive is ~700MB, the file-system as part of Vision-SDK release is ~60MB. This reduction in size is achieved by removing components not required for traditional ADAS use-cases.

Follow the instructions below to rebuild the Vision-SDK file-system:

1. Build the Yocto file-system by following the instructions as part of the Processor_SDK_Linux_Automotive_Software_Developers_Guide wiki.
2. Apply the patches present in the linux-kernel-addon/fs-patches/yocto/meta-glsdk folder of Vision-SDK to the tisdk/sources/meta-glsdk folder in the yocto repository.
3. Apply the patches present in the linux-kernel-addon/fs-patches/yocto/meta-arago folder of Vision-SDK to the tisdk/sources/meta-arago folder in the yocto repository.
4. Rebuild the file-system by running the bitbake command as documented in the Processor_SDK_Linux_Automotive_Software_Developers_Guide wiki.
5. The changes in meta-arago are for the reduction in file-system size while the changes in meta-glsdk are for the VDRM and VDRM+ IPUMM-based decode.

9 References

- Processor SDK Linux Automotive Software Developers Guide

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (https:www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265